

PATENT OFFICE  
JAPANESE GOVERNMENT

This is to certify that the annexed is a true copy of the following application  
as filed with this Office.

Date of Application : December 28, 1999

Application Number : P11-375859

Applicant(s): TOSHIBA LSI SYSTEM SUPPORT KABUSHIKI KAISHA  
KABUSHIKI KAISHA TOSHIBA

December 22, 2000

Commissioner,  
Patent Office

Kouzou OIKAWA

Number of Certificate: 2000-3105725

日本国特許庁  
PATENT OFFICE  
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されて  
る事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed  
this Office.

願年月日  
Date of Application:

1999年12月28日

願番号  
Application Number:

平成11年特許願第375859号

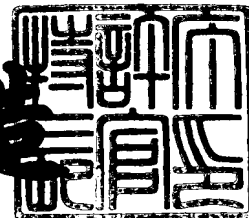
願人  
Applicant(s):

東芝エルエスアイシステムサポート株式会社  
株式会社東芝

2000年12月22日

特許庁長官  
Commissioner,  
Patent Office

及川耕造



CERTIFIED COPY OF  
PRIORITY DOCUMENT

出証番号 出証特2000-3105725

BEST AVAILABLE COPY

【書類名】 特許願

【整理番号】 46A999500

【提出日】 平成11年12月28日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 11/32

【発明の名称】 情報処理装置、及び不具合解析プログラムを格納した記録媒体

【請求項の数】 9

【発明者】

    【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 番地 株式会社東芝  
                                マイクロエレクトロニクスセンター内

    【氏名】 山内 信之

【発明者】

    【住所又は居所】 神奈川県川崎市幸区堀川町 5 8 0 番地 東芝エルエスアイシステムサポート株式会社内

    【氏名】 松本 奈津美

【特許出願人】

    【識別番号】 598010562

    【氏名又は名称】 東芝エルエスアイシステムサポート株式会社

【特許出願人】

    【識別番号】 000003078

    【氏名又は名称】 株式会社 東芝

【代理人】

    【識別番号】 100083806

    【弁理士】

    【氏名又は名称】 三好 秀和

    【電話番号】 03-3504-3075

【選任した代理人】

    【識別番号】 100068342

【弁理士】

【氏名又は名称】 三好 保男

【選任した代理人】

【識別番号】 100100712

【弁理士】

【氏名又は名称】 岩▲崎▼ 幸邦

【選任した代理人】

【識別番号】 100100929

【弁理士】

【氏名又は名称】 川又 澄雄

【選任した代理人】

【識別番号】 100108707

【弁理士】

【氏名又は名称】 中村 友之

【選任した代理人】

【識別番号】 100095500

【弁理士】

【氏名又は名称】 伊藤 正和

【選任した代理人】

【識別番号】 100101247

【弁理士】

【氏名又は名称】 高橋 俊一

【選任した代理人】

【識別番号】 100098327

【弁理士】

【氏名又は名称】 高松 俊雄

【手数料の表示】

【予納台帳番号】 001982

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 情報処理装置、及び不具合解析プログラムを格納した記録媒体

【特許請求の範囲】

【請求項 1】 プログラムの実行履歴情報をもとに、該プログラムの動作状況を時系列にユーザーに表示する表示手段と、

表示された動作状況の中でユーザーが不具合の箇所を指定するための入力手段と、

前記入力手段によってユーザーから指定された不具合の箇所と該プログラムの動作状況から該不具合要因を解析し、この不具合要因を解決するための解決策を特定する動作解析手段と、

を有し、

前記動作解析手段は、前記解決策を反映させた前記動作状況を再作成し、

前記表示手段は、前記不具合要因と前記解決策と再作成した動作状況とをユーザーに表示する、

ことを特徴とする情報処理装置。

【請求項 2】 前記動作解析手段において、前記解決策を反映させる箇所を特定することができない場合に、

更に、ユーザーに該解決策を反映させる箇所の指定を促し、

前記動作解析手段は、このユーザーにより指定された箇所に該解決策を反映させた前記動作状況を再作成し、

前記表示手段は、前記不具合要因と解決策と再作成した動作状況とをユーザーに表示する、

ことを特徴とする請求項 1 記載の情報処理装置。

【請求項 3】 前記情報処理装置は、前記解決策を特定するためのプログラム機能対応情報を有することを特徴とする請求項 1 ないし請求項 2 のいずれかに記載の情報処理装置。

【請求項 4】 前記入力手段は、ユーザーが不具合の箇所を座標位置により指定することで、該指定箇所を一意に認識することを特徴とする請求項 1 ないし請求項 3 のいずれかに記載の情報処理装置。

【請求項 5】 前記情報処理装置は、更に、

前記解決策と前記実行履歴情報とをもとに、本来の仕様を満たすプログラムのスケルトンを自動生成することを特徴とする請求項 1 ないし請求項 4 のいずれかに記載の情報処理装置。

【請求項 6】 プログラムの実行履歴情報をもとに、該プログラムの動作状況を時系列にユーザーに表示するステップと、

ユーザーから指定された不具合の箇所と該プログラムの動作状況から該不具合要因を解析し、この不具合要因を解決するための解決策を特定するステップと、

前記解決策を反映させた前記動作状況を再作成するステップと、

前記不具合要因と前記解決策と再作成した動作状況とをユーザーに表示するステップと、

を有することを特徴とする不具合解析プログラムを格納した記録媒体。

【請求項 7】 前記解決策を反映させる箇所を特定することができない場合に、ユーザーに該解決策を反映させる箇所の指定を促すステップと、

この指定された箇所に該解決策を反映させた前記動作状況を再作成するステップと、

前記不具合要因と前記解決策と再作成した動作状況とをユーザーに表示するステップと、

を有することを特徴とする請求項 6 記載の不具合解析プログラムを格納した記録媒体。

【請求項 8】 前記不具合解析プログラムは、前記解決策を特定するためのプログラム機能対応情報テーブルを有することを特徴とする請求項 6 ないし請求項 7 のいずれかに記載の不具合解析プログラムを格納した記録媒体。

【請求項 9】 前記不具合解析プログラムは、更に、

前記解決策と前記実行履歴情報とをもとに、本来の仕様を満たすプログラムのスケルトンを自動生成するステップを有することを特徴とする請求項 6 ないし請求項 8 のいずれかに記載の不具合解析プログラムを格納した記録媒体。

【発明の詳細な説明】

【 0 0 0 1 】

【発明の属する技術分野】

本発明は、リアルタイムOS (Real-Time Operating System) 等で動作するマルチタスク・プログラムのテストやデバッグなどに利用される情報処理装置、及び不具合解析プログラムを格納した記録媒体に関する。

【0002】

【従来の技術】

TRON (The Real-time Operating system Nucleus) に代表されるリアルタイムOSの適用範囲は、マイクロプロセッサ技術の発展により拡大の一途をたっており、産業用途のみならず、通信機器やオフィス機器などの業務機器分野、家電や携帯電話などの民生用機器などへも急激に拡大している。

【0003】

一般に、このようなリアルタイムOS配下で実行されるマルチタスク方式のアプリケーション・プログラムの不具合解析をするために、各プログラムの動きを追跡するのは容易でない。リアルタイムOSでは、特定のタイミングで動作するタスクを次々と切り替えていくことによって、複数のプログラムを動かすマルチタスク方式が採られているため、一つのプログラムを追うだけでは解析できないからである。

【0004】

そのため、プログラムの動作を追跡する機能を有するOSが開発され、プログラムの動作状況を解析するためのひとつの手掛りとして、プログラムの実行履歴情報が用いられている。図27に示すように、プログラムの実行履歴情報4は、プログラム動作の追跡機能を有するOS101によってOS101のメモリ上に出力されたり、更にデバッガ102等の開発ツールによってデバッガ102等が持つメモリ上に出力されたりする。

【0005】

図29に、μITRON (Micro Industrial TRON) 仕様のプログラム実行履歴情報を例示する。図29(a)は、システムコール発行履歴のフォーマット例であり、1レコードで1事象を表し、機能のタイプ (type)、発行元タスク (oid)、システムコールの種類 (sysid)、発行先タスク (obj) で構成されてい



る。出力される実行履歴情報の種類としては、このようなシステムコール発行履歴の他に、タスク切り換え履歴、ハンドラ実行履歴などがある。

#### 【0006】

図29（b）は、OS101やデバッガ102等によってメモリ上に出力された実行履歴データの例である。（イ）は、タスク切り換えが行われたことを表しており、アイドル状態（タスク＝0）からタスク1（タスク＝1）に制御が移行している。（ロ）は、システムコールが発行されたことを表しており、タスク1（oid=1）からsta\_tsk（sysid=-19）という種類のシステムコールがタスク2（obj=2）に対して発行されている。

#### 【0007】

#### 【発明が解決しようとする課題】

上述したように、プログラムの実行履歴をデータとして保存することができるようになり、プログラムの不具合を解析するための手掛りとして利用されているが、以下のような問題がある。

#### 【0008】

第1に、コントローラのメモリ容量の関係から、履歴情報を採取できる量には制限があり、実際には図28のように、履歴情報を細切れに保存することになってしまう。特に、ローコストを目指したシステムの場合、このメモリ容量はかなり限定されてしまう。

#### 【0009】

第2に、出力される履歴情報は、図29に示したように、主に数字からなる文字の羅列であり、しかも大量に出力されがちであることから、この履歴情報を手掛りにプログラムの動作軌跡を辿って不具合を解析することは、大変困難である。

#### 【0010】

その結果、プログラムの不具合解析には、依然として多大な時間を要し、開発コストや開発期間などに大きな影響を与えていた。

#### 【0011】

本発明はこのような課題を解決するためになされたものであって、対話形式で

ユーザーから指摘された不具合箇所とプログラムの実行履歴情報から、不具合要因と解決策を特定し、これをユーザーに提示することができる情報処理装置、及び情報処理装置に搭載される不具合解析プログラムを格納した記録媒体を提供することを目的とする。

【 0 0 1 2 】

本発明の第 2 の目的は、前記特定した不具合要因と解決策から、該不具合を解決したプログラムを生成してユーザーに提示することができる情報処理装置、及び情報処理装置に搭載される不具合解析プログラムを格納した記録媒体を提供することを目的とする。

【 0 0 1 3 】

【課題を解決するための手段】

上記課題を解決するために、本発明は、プログラムの実行履歴情報をもとに、該プログラムの動作状況を時系列にユーザーに表示する表示手段と、表示された動作状況の中でユーザーが不具合の箇所を指定するための入力手段と、前記入力手段によってユーザーから指定された不具合の箇所と該プログラムの動作状況から該不具合要因を解析し、この不具合要因を解決するための解決策を特定する動作解析手段と、を有し、前記動作解析手段は、解決策を反映させた前記動作状況を再作成し、前記表示手段は、前記不具合要因と前記解決策と再作成した動作状況とをユーザーに表示する、ことを特徴とする。このとき、前記解決策を反映させる箇所を特定することができない場合には、更に、ユーザーによって該解決策を反映させる箇所を指定させ、前記動作解析手段は、この指定された箇所に該解決策を反映させた前記動作状況を再作成し、前記表示手段は、前記不具合要因と解決策と再作成した動作状況とをユーザーに表示することを特徴とする。

【 0 0 1 4 】

更にまた、前記解決策と前記実行履歴情報とをもとに、本来の仕様を満たすプログラムのスケルトンを自動生成することを特徴とする。

【 0 0 1 5 】

本発明により、対話形式により、ユーザーから指摘された不具合箇所とプログラムの実行履歴情報から不具合要因と解決策を特定し、これをユーザーに提示す

ることができる。

【0016】

また、前記特定した不具合要因と解決策から、該不具合を解決したプログラムを生成して同様にユーザーに提示することができる。

【0017】

【発明の実施の形態】

以下、図面に基づいて、本発明の実施形態について説明する。

【0018】

図1は本発明に係る情報処理装置の一実施形態を示す概略図であり、図2は本発明に係る情報処理装置に搭載される不具合解析プログラムの処理の流れを示した流れ図である。

【0019】

図1において、この情報処理装置は、ユーザー5からの入力11を受付け、又ユーザー5に情報を提示12するユーザーインタフェース部1と、ユーザーインタフェース部1が受付けたユーザー5からの入力情報を解析してタスク動作判定部3に送る要求解析部2と、実行履歴ファイル4に格納されているプログラムの実行履歴情報11とシステムコール対応テーブル7とタスク動作判定部3からの情報をもとにプログラムの不具合要因とその解決策を特定し、各種解析結果情報をユーザーインタフェース部1に送り、又解析結果情報から不具合を解決するプログラムを生成してプログラムファイルに出力するタスク動作判定部3とから構成される。

【0020】

続いて、図2をもとに、全体の処理の流れを説明する。

【0021】

ユーザーインタフェース部1は、予めタスク動作判定部3がプログラムの動作履歴情報11をもとに作成した検証データ14（図3）をユーザー5に表示する（ST01）。ユーザー5は表示された検証データ14を見て、動作的に不具合がないと確認した場合、処理を終了する（ST04）。一方、ユーザー5は表示された検証データ14の中に不具合を発見した場合、その不具合箇所を指摘する（ST05）

。ユーザーインタフェース部 1 は、ユーザー 5 が入力した不具合の箇所 1 2 を、要求解析部 2 に転送する (ST06)。

#### 【0022】

次に、要求解析部 2 は、ユーザーインタフェース部 1 から受取ったユーザー 5 が入力した指摘箇所 1 2 と先にタスク動作判定部 3 が作成した検証データ 1 4 から、比較データ 1 3 (図 4) を作成し (ST07)、作成した比較データ 1 3 をタスク動作判定部 3 に転送する (ST08)。

#### 【0023】

次に、タスク動作判定部 3 は、要求解析部 2 から受取った比較データ 1 3 と先に作成しておいた検証データ 1 4 とを比較して、要求タスクの状態を検索し (ST09)、不具合の要因を解析する (ST10)。タスク動作判定部 3 は不具合の要因を特定したら、不具合を解決する機能を検証データ 1 4 に反映させた上で、検証データ 1 4 をユーザーインタフェース部 1 に転送する (ST11)。続いて、タスク動作判定部 3 は、判定結果データ 1 (図 5) 及び判定結果データ 2 (図 6) を作成してユーザーインタフェース部 1 に転送する (ST12)。

#### 【0024】

ここまでで不具合が全て解決されていて (ST13)、且つ処理終了をする場合 (ST15) には、ST11で更新した検証データ 1 4 をもとに不具合を解決した正常プログラム・ソース 1 8 を生成し (ST16)、ユーザーインタフェース部 1 に制御を戻す。一方、未解決の不具合要因が残っている場合 (ST14)、不具合を解決するためにユーザーに問い合わせる不具合解決用質問データ 1 7 (図 7) を作成し、ユーザーインタフェース部 1 に転送し (ST14)、ユーザーインタフェース部 1 に制御を戻す。

#### 【0025】

以上説明したST01からST16までの処理を、プログラムの不具合が解決されて処理を終了するまで繰り返す。

#### 【0026】

次に、実際に不具合が存在するプログラム例に基づいて、本発明の不具合解析プログラムを搭載した情報処理装置の処理動作について、更に詳細に説明する。

【0027】

ここでは、 $\mu$ ITRON 3.0仕様のリアルタイムOS配下で動作するアプリケーション・プログラムを例に説明する。

【0028】

(第1実施例)

本実施例では、アラーム機能付きの時計のプログラムを想定する。本プログラムで使用するタスクは以下の3本である。

【0029】

(1) startup (task\_ID=1,priority=1) [アラーム設定モード]

(2) タスク A (task\_ID=2,priority=3) [ノーマルモード]

(3) タスク B (task\_ID=3,priority=2)

本プログラムは、アラームを設定すると(アラーム設定モードを処理すると)、ノーマルモードに移行し、その後処理を終了するものであり、本来は図8に示すタイムチャートのような動作を想定して作成されたアプリケーション・プログラムである。従って、例えば処理終了時点を注目してみると、タスク A (ノーマルモード)で処理が終了する、つまりその時点での最終実行タスクはタスク A である、というのがこのプログラムの本来の仕様である。

【0030】

ところが、実際にこのプログラムを動作させてみたところ、図9に示すような結果が得られたとする。実行結果の処理終了時点を注目してみると、startup タスク(アラーム設定モード)で処理が終了している。すなわち、このプログラムは本来の仕様を満たしていない、このプログラムには不具合が潜在しているということである。

【0031】

このような状況において、ユーザー 5 は本発明の情報処理装置及び情報処理装置に搭載された不具合解析プログラムを使用して、以下のように不具合を解決することができる。

【0032】

[処理 1]

ユーザーインタフェース部 1 の処理の流れを図 1 0 に示す。

【 0 0 3 3 】

ユーザーインタフェース部 1 は、予めタスク動作判定部 3 がプログラムの動作履歴情報 1 1 をもとに作成した検証データ 1 4 (図 3) をユーザー 5 に表示する (ST21~ST24) 。ユーザー 5 への表示例を図 1 1 に示す。図 1 1 のようにプログラムの各タスクの動作とタスクの切り換えの様子を時系列に表せば、ユーザー 5 は表示されたプログラムの動作の中の不具合を指摘しやすい。特に、数字の羅列である実行履歴情報 1 1 をグラフィックで表現することは有効である。

【 0 0 3 4 】

ここで、ユーザー 5 は当該プログラムの不具合箇所を指摘する入力を行う (ST 26) 。この場合、図 1 1 に示した位置 (最終イベントの後のタスク A の位置) にマウскарソルを指定したとする。この指定位置がユーザー 5 が指摘した不具合の箇所となる。本来のプログラムの仕様では、この時点ではタスク A は実行状態でなければならないはずである。

【 0 0 3 5 】

ユーザーインタフェース部 1 は、ユーザー 5 が入力した不具合の箇所データ 1 2 を、要求解析部 2 に転送 (ST27) し、要求解析部 2 に制御を移す。このとき転送する不具合箇所データ 1 2 は、図 1 1 においてマウскарソルを指定した位置、すなわちイベント順を示していた X 軸とタスクを示していた Y 軸との座標位置で一意に認識することができる。ユーザーが指摘した箇所データ 1 2 は、X = 7 , Y = 3 である。

【 0 0 3 6 】

続いて、図 1 2 に要求解析部 2 の処理の流れを示す。

【 0 0 3 7 】

要求解析部 2 は、ユーザーインタフェース部 1 から受取った不具合箇所データ 1 2 と先にタスク動作判定部 3 が作成した検証データ 1 4 から、要求内容を解析する。まず、X 座標値と検証データ 1 4 のイベントとを比較し、ユーザー 5 の要求タイミングを特定する (ST28) 。次に、Y 座標値と検証データ 1 4 のアイテム順から、ユーザーが要求しているタスクを特定する (ST29) 。このときの処理イ

メッセージを図 1 3 に示す。すなわち、ユーザー 5 の要求タイミングはイベントの最後（第 7 イベントの位置）、要求タスクはタスク A（3 番目のアイテム）である。

【 0 0 3 8 】

要求解析部 2 は、このようにして比較データ 1 3 を作成し（ST30）、作成した比較データ 1 3 をタスク動作判定部 3 に転送して、制御をタスク動作判定部 3 に移す。尚、このときの検証データ 1 4 と比較データ 1 3 を図 1 6 に示す。

【 0 0 3 9 】

〔第 1 次要求〕

この場合の要求タスクはタスク A であり、要求タイミングはイベントの最後である。

【 0 0 4 0 】

〔処理 2〕

続いて、図 1 4 にタスク動作判定部 3 の処理の流れを示す。

【 0 0 4 1 】

タスク動作判定部 3 は、要求解析部 2 から受取った比較データ 1 3 と先に作成しておいた検証データ 1 4 とを比較して、要求タスクの状態を検索する（ST31）。図 1 6 に示す検証データ 1 4 を検索してみると、イベントの最後の時点でのタスク A の状態は、「実行可能（READY）」状態であることが判る（ST32）。「実行可能」状態にあるタスクとは、実行する準備は整っているが、自分より優先度の高いタスクもしくは優先度が自分と同じであるタスクが「実行（RUNNING）」状態であるために、実行できない状態にある、言いかえると、実行できる状態のタスクの中で自分が最高の優先順位になればいつでも実行できる状態にあるタスクである。

【 0 0 4 2 】

本発明の情報処理装置の不具合解析処理が終了するのは（目的を達成するのは）、ユーザーが第 1 次要求で指摘したタスク、すなわちタスク A がイベントの最後の時点で「実行」状態になることである。

【 0 0 4 3 】

タスク A を「実行」状態にするまで（不具合要因を特定するまで）、以下に説明する判定結果データ 1、判定結果データ 2 を作成し、作成した判定結果データから本来あるべき処理を特定し、それを検証データ 1 4 に反映させる。このときの各データの状態を図 1 7 に示す（但し、図 1 7 は処理完了状態の図である）。

【0 0 4 4】

まず、N の値を判定結果データ 1 の①と判定結果データ 2 の①に入れる（ST33）。ここでは第 1 次要求であるため、N の値は「1」である。それぞれの欄に「1」が入る。

【0 0 4 5】

次に、判定結果データ 1 の②に要求タスクであるタスク A を、③にタスク A の状態を入れる（ST34）。

【0 0 4 6】

次に、検証データ 1 4 から、イベントの最後の時点で「実行」であったタスクを検索する（ST35）。この場合、イベントの最後の時点で「実行」であったタスクは、startup タスクである。検索したこのタスクを判定結果データ 2 に入れる（ST36）。

【0 0 4 7】

次に、判定結果データ 2 に入れたタスク（この場合、startup タスク）が ext\_tsk を発行した場合の検証データ 1 4 を作成する（ST37）。具体的には、7 番目のイベントに、startup タスクが ext\_tsk を発行するというイベントを検証データ 1 4 に作成するのである。（必然的に、第 8 イベントとしてタスク切り換え（Task Dispatch）が発生する。）

そして、このイベント（第 7 ～ 8 イベント）を検証データ 1 4 に反映させたことで、第 1 次要求のタスク A が「実行」状態に遷移したかどうかを、検証データ 1 4 から検索するのである（ST38）。

【0 0 4 8】

以上の ST33 ～ ST39 までの処理を、第 1 次要求のタスク A が「実行」状態になるまで繰り返す（ST39）。

【0 0 4 9】



この場合に、「実行」状態に遷移したのは、タスク A ではなく、タスク B である。従って、再び ST33 に戻って、タスク A がイベントの最後の時点で「実行」状態になるための処理を行う。

【0 0 5 0】

〔処理 3〕

まず、N の値を判定結果データ 1 の①と判定結果データ 2 の①に入れる（ST33）。ここでは第 1 次要求であるため、N の値は「1」である。それぞれの欄に「1」が入る。

【0 0 5 1】

次に、判定結果データ 1 の②に要求タスクであるタスク A を、③にタスク A の状態を入れる（ST34）。検証データ 1 4 から、タスク A の状態は「実行可能」であることが判る。

【0 0 5 2】

次に、検証データ 1 4 から、イベントの最後の時点で「実行」状態であったタスクを検索する（ST35）。この場合、イベントの最後の時点で「実行」状態であったタスクはタスク B である。検索したこのタスク B を判定結果データ 2 に入れる（ST36）。

【0 0 5 3】

次に、判定結果データ 2 に入れたタスク（この場合、タスク B）が ext\_tsk を発行した場合の検証データ 1 4 を作成する（ST37）。具体的には、9 番目のイベントに、タスク B が ext\_tsk を発行するというイベントを検証データ 1 4 に作成するのである。（必然的に、第 10 イベントとしてタスク切り換え（Task Dispatch）が発生する。）

そして、このイベント（第 9～10 イベント）を検証データ 1 4 に反映させたことで、第 1 次要求のタスク A が「実行」状態に遷移したかどうかを、検証データ 1 4 から検索するのである（ST38）。

【0 0 5 4】

この場合は、タスク A が「実行」の状態に遷移したので、処理終了のフラグ等をセットして（ST51）、イベント 7～10 を反映させた検証データ 1 4 から今回

の不具合を解決したプログラムのスケルトンを生成し、プログラムファイル 6 に出力する（ST52）。プログラムの生成例を図 1 8 に示す。図 1 8 の（a）及び（b）が、今回の不具合要因とその解決策であった。

【0 0 5 5】

図 1 7 に示すように、第 1 次要求のタスク A が「実行」状態に至るまで要した判定結果データ 1，2 と検証データ 1 4 を、ユーザーインタフェース部 1 に転送し、制御をユーザーインタフェース部 1 に戻す。

【0 0 5 6】

ユーザーインタフェース部 1 は、タスク動作判定部 3 から転送された検証データ 1 4、判定結果データ 1，2 をユーザーに表示する。これにより、今回の不具合の要因とその解決策をユーザー 5 に提示することができ、また、本来の仕様に沿ったプログラムの動作をユーザー 5 に提示することができる。

【0 0 5 7】

また、本来の仕様を満たしたプログラムがプログラムファイル 6 に出力されているので、ユーザー 5 はこれを利用することができる。

【0 0 5 8】

（第 2 実施例）

本実施例で想定するプログラムが使用するタスクは以下の 4 本である。

【0 0 5 9】

- （1）startup （task\_ID=1,priority=1）
- （2）タスク A （task\_ID=2,priority=2）
- （3）タスク B （task\_ID=3,priority=2）
- （4）タスク C （task\_ID=4,priority=2）

（使用するセマフォ：Semaphore（ID=1,初期セマフォカウンタ=0））

本プログラムは、第 1 実施例と同様に、本来は図 1 9 に示すタイムチャートのような動作を想定して作成されたアプリケーション・プログラムである。従って、例えば処理終了時点を注目してみると、タスク B で処理が終了する、つまりその時点での最終実行タスクはタスク B である、というのがこのプログラムの本来の仕様である。

【0060】

ところが、実際にこのプログラムを動作させてみたところ、図20に示すような結果が得られたとする。実行結果によると、Idle Mode（レディーキューにタスクがない状態）になっている。

【0061】

このような状況において、ユーザー5は本発明の情報処理装置及び情報処理装置に搭載された不具合解析プログラムを使用して、以下のように不具合を解決することができる。

【0062】

〔処理1〕

図10のフローチャートに沿って、処理の流れを説明する。

【0063】

ユーザーインタフェース部1は、予めタスク動作判定部3がプログラムの動作履歴情報11をもとに作成した検証データ14をユーザー5に表示する（ST21～ST24）。

【0064】

ここで、ユーザー5は不具合箇所を指摘する入力を行う（ST26）。この場合、実施例1と同様に、ユーザー5はイベントの最後を指摘したものとする。この指定位置がユーザー5が指摘した不具合の箇所となる。本来のプログラムの仕様では、この時点ではタスクBは実行状態でなければならないはずである。

【0065】

ユーザーインタフェース部1は、ユーザー5が入力した不具合の箇所データ12を、要求解析部2に転送（ST27）し、要求解析部2に制御を移す。ユーザーが指摘した箇所データ12は、X=12，Y=4である。

【0066】

要求解析部2の処理の流れを図12のフローチャートに沿って説明する。

【0067】

要求解析部2は、ユーザーインタフェース部1から受取った不具合箇所データ12と先にタスク動作判定部3が作成した検証データ14から、要求内容を解析

する。まず、X座標値と検証データ 1 4 のイベントとを比較し、ユーザー 5 の要求タイミングを特定する (ST28)。次に、Y座標値と検証データ 1 4 のアイテム順から、ユーザーが要求しているタスクを特定する (ST29)。この場合、ユーザー 5 の要求タイミングはイベントの最後 (第 1 2 イベントの位置)、要求タスクはタスク B (4 番目のアイテム) である。

【0 0 6 8】

要求解析部 2 は、このようにして比較データ 1 3 を作成し (ST30)、作成した比較データ 1 3 をタスク動作判定部 3 に転送して、制御をタスク動作判定部 3 に移す。尚、このときの検証データ 1 4 と比較データ 1 3 を図 2 1 に示す。

【0 0 6 9】

[第 1 次要求]

この場合のユーザー 5 の要求タスクはタスク B であり、要求タイミングはイベントの最後である。

【0 0 7 0】

[処理 2]

続いて、タスク動作判定部 3 の処理の流れを図 1 4 のフローチャートに沿って説明する。

【0 0 7 1】

タスク動作判定部 3 は、要求解析部 2 から受取った比較データ 1 3 と先に作成しておいた検証データ 1 4 とを比較して、要求タスクの状態を検索する (ST31)。図 2 1 に示す検証データ 1 4 を検索してみると、要求タイミングである「イベントの最後」の時点でのタスク B の状態は、「待ち (WAITING)」状態であることが判る (ST32、ST40)。「待ち」状態にあるタスクとは、そのタスクを実行できる何らかの条件が整わないために実行できない状態にある、言いかえると、何らかの条件が満たされるのを待っている状態にあるタスクである。

【0 0 7 2】

本発明の情報処理装置の不具合解析処理が終了するのは (目的を達成するのは)、ユーザーが第 1 次要求で指摘したタスク、すなわちタスク B がイベントの最後の時点で「実行」状態になることである。

【 0 0 7 3 】

この場合には、タスク B の状態を「実行」状態にする前に、まず「実行可能」状態にする必要がある。「待ち」状態にあるタスクは、直接「実行」状態には移行できないためである。その処理を以下に説明する。

【 0 0 7 4 】

まず、N の値を判定結果データ 1 の①と不具合解決用質問データの①に入れる (ST45)。ここでは第 1 次要求であるため、N の値は「1」である。それぞれの欄に「1」が入る。

【 0 0 7 5 】

次に、判定結果データ 1 の②に要求タスクであるタスク B を、③にタスク B の状態（「待ち」）を入れる (ST46)。

【 0 0 7 6 】

次に、この第 1 次要求の要求タスク B を「実行可能」状態に遷移させることができるシステムコールをシステムコール対応テーブル 7 から検索する (ST47)。このシステムコール対応テーブル 7 の一例を図 1 5 に示す。

【 0 0 7 7 】

この場合、図 2 1 の検証データ 1 4 によると、タスク B は「slp\_tsk」によって「（起床）待ち」の状態にあるため、図 1 5 のシステムコール対応テーブル 7 から「slp\_tsk」に対応するシステムコールを検索するのである。図 1 5 のシステムコール対応テーブル 7 を検索すると、2 番目のエントリに「slp\_tsk」があり、それに対応するシステムコールは「wup\_tsk」である。この「wup\_tsk」を発行すれば、タスク B の状態を「待ち」から「実行可能」に遷移させることができるのである。検索結果の「wup\_tsk」を不具合解決用質問データの③に入れる (ST48)。

【 0 0 7 8 】

次に、不具合解決用質問データの②にタスク B の状態を「待ち」から「実行可能」に遷移させるシステムコールである「wup\_tsk」を発行対象タスクを入れる (ST49)。この場合、発行対象タスクはタスク A である。

【 0 0 7 9 】

この「wup\_tsk」を「何時」「何処」で発行するか、ということは機械的には決められない。発行するタイミング、発行する場所によって、処理が変わってしまうからである。従って、このシステムコール「wup\_tsk」を「何時」「何処」で発行すべきなのかをユーザー 5 に問合せる。この問合せのためのデータが不具合解決用質問データなのである。

【0080】

この不具合解決用質問データによる問合せを第 2 次要求とする（N に 1 加算する（ST50））。ユーザーインタフェース部 1 に制御を戻し、ユーザー 5 に検証データと不具合解決用質問データを提示し、質問に対する回答の入力を促す。

【0081】

[第 2 次要求]

この場合、ユーザー 5 が要求した要求タスクはタスク A、要求タイミングは「システムコール「wai\_sem」を発行した後」であったとする。

【0082】

[処理 3]

ユーザーインタフェース部 1 に戻って、ユーザー 5 が入力した箇所のデータ 1 2 を、要求解析部 2 に転送（ST27）し、要求解析部 2 に制御を移す。このとき転送する箇所データ 1 2 は、 $X = 6$ ， $Y = 3$  である。

【0083】

要求解析部 2 は、比較データ 1 3 を作成し（ST30）、作成した比較データ 1 3 をタスク動作判定部 3 に転送して、制御をタスク動作判定部 3 に移す。尚、このときの検証データ 1 4 と比較データ 1 3 を図 2 2 に示す。

【0084】

タスク動作判定部 3 は、要求解析部 2 から受取った比較データ 1 3 と検証データ 1 4 とを比較して、要求タスクの状態を検索する（ST31）。図 2 2 に示す検証データ 1 4 を検索してみると、要求タイミングでのタスク A の状態は、「（セマフォ資源の獲得）待ち」状態であることが判る（ST32、ST40）。この場合には、タスク A の状態を「待ち」から「実行可能」にする必要がある。その処理を以下に説明する。

【0085】

まず、Nの値を判定結果データ1の①と不具合解決用質問データの①に入れる(ST45)。ここでは第2次要求であるため、Nの値は「2」である。それぞれの欄に「2」が入る。

【0086】

次に、判定結果データ1の②に要求タスクであるタスクAを、③にタスクAの状態(「待ち」)を入れる(ST46)。

【0087】

次に、この第2次要求の要求タスクAを「実行可能」状態に遷移させることができるシステムコールをシステムコール対応テーブル7から検索する(ST47)。対応するシステムコールは「sig\_sem」である。この「sig\_sem」を発行すれば、タスクBの状態を「待ち」から「実行可能」に遷移させることができるのである。検索結果の「sig\_sem」を不具合解決用質問データの③に入れる(ST48)。

【0088】

次に、不具合解決用質問データの②にタスクBの状態を「待ち」から「実行可能」に遷移させるシステムコールである「sig\_sem」を発行対象タスクを入れる(ST49)。この場合、発行対象タスクはタスクCである。

【0089】

この「wup\_tsk」を「何時」「何処」で発行するか、ということも機械的には決められない。発行するタイミング、発行する場所によって、処理が変わってしまうからである。従って、このシステムコール「sig\_sem」を「何時」「何処」で発行するべきなのかをユーザー5に問合せる。

【0090】

この不具合解決用質問データによる問合せを第3次要求とする。Nに1加算して、ユーザーインタフェース部1に制御を戻し(ST50)、ユーザー5に検証データと不具合解決用質問データを提示し、質問に対する回答の入力を促す。

【0091】

[第3次要求]

この場合、ユーザー5が要求した要求タスクはタスクC、要求タイミングは「

システムコール「ext\_tsk」を発行する前」であったとする。

【0092】

〔処理4〕

ユーザーインタフェース部1に戻って、ユーザー5が入力した箇所のデータ12を、要求解析部2に転送（ST27）し、要求解析部2に制御を移す。

【0093】

要求解析部2は、比較データ13を作成し（ST30）、作成した比較データ13をタスク動作判定部3に転送して、制御をタスク動作判定部3に移す。尚、このときの検証データ14と比較データ13を図23に示す。

【0094】

タスク動作判定部3は、要求解析部2から受取った比較データ13と検証データ14とを比較して、要求タスクの状態を検索する（ST31）。図22に示す検証データ14を検索してみると、要求タイミング「システムコール「ext\_tsk」を発行する前」時点でのタスクCの状態は、「実行」状態であることが判る（ST32、ST40）。

【0095】

このように要求タスクの要求タイミング時点での状態が「実行」状態である場合には、ひとつ前の（N-1の）不具合解決用質問データのシステムコールを要求タイミング時に発行した場合の検証データ14を作成する（ST41）。本実施例でいうと、不具合解決用質問データのN=2の時のシステムコールである「sig\_sem」をタスクCが発行した場合の検証データを作成することになる（図24の検証データを参照）。

【0096】

次に、第2次要求タスクのイベントの最後の時点での状態を検証データ14から検索する（ST42、ST43）。

【0097】

検索の結果、第2次要求タスクAは「実行」状態に遷移している。判定結果データ1のタスクAの状態を「実行」状態に書きかえる（ST44）。

【0098】



これで、第 2 次要求タスクの状態が「実行」状態に遷移したことで、第 3 次要求は満たされた。しかし、第 1 要求のタスク B は依然として「実行可能」状態のままであるため、ここまで作成した検証データ、判定結果データ、不具合解決用質問データ（第 1 次分）をユーザーインタフェース部 1 に転送し、制御をユーザーインタフェース部 1 に戻す（ST38～39）。

【0099】

そして、これまで説明した処理に準じて、第 1 次要求のタスク B が第 1 次要求タイミングである「イベントの最後」の時点で「実行」状態に遷移するまで、処理を繰り返す。このタスク B が「イベントの最後」の時点で「実行」状態に遷移するまでの検証データ、比較データ、判定結果データ 1、判定結果データ 2、及び不具合解決用データの状態を、図 2 1～図 2 6 に示す。

【0100】

図 2 6 に示した検証データが本来のプログラムの仕様を満たすものであり、同じく図 2 6 に示した判定結果データ 1、2 と不具合解決質問データが今回の不具合の要因となっていたのもである。本発明の情報処理装置は、これらの各データをユーザー 5 に提示することで、プログラムの不具合要因とその解決策を対話形式でユーザーに提示できる。

【0101】

また、処理終了時には、第 1 実施例と同様にして、図 2 6 の検証データからプログラムのスケルトンを自動生成し、ユーザー 5 に提供することができる。

【0102】

以上、第 1 実施例、第 2 実施例で説明したように、本発明の不具合解析プログラムを搭載した情報処理装置を用いることにより、対話形式で不具合要因とその解決策を容易に特定でき、更に、本来の仕様に沿ったプログラムの動作、及び本来の仕様に沿ったプログラムをも容易に取得することができる。

【0103】

以上、本発明について、詳細に説明したが、本発明は本実施例に限定されず、本発明の主旨を逸脱しない範囲において、種々の改良や変更を成し得るであろう。例えば、本実施例では、イベントの最後（プログラムの処理終了時点）のタス

クの状態に着目して不具合要因を解析する例を示したが、この着目するポイントはこれに限定されず、ユーザーは任意のポイントに着目して不具合要因の解析することができる。

【0 1 0 4】

また、本実施例では I T R O N に準拠したアプリケーションプログラムを例に説明したが、本発明はこのようリアルタイム O S に限定されず、一般にマルチタスク方式で動作するプログラムに広く適用できる。

【0 1 0 5】

従って、本発明はこの開示から妥当な特許請求の範囲に係わる発明特定事項によってのみ限定されるものでなければならない。

【0 1 0 6】

【発明の効果】

本発明によれば、対話形式でユーザーから指摘された不具合箇所とプログラムの実行履歴情報から不具合要因とその解決策を特定し、更にこの解決策を反映させたプログラムの動作をユーザーに提示することができる。

【0 1 0 7】

また、前記特定した不具合要因とその解決策から、該不具合を解決したプログラムを自動生成してユーザーに提示することができる。

【0 1 0 8】

その結果、プログラムの不具合解析作業を容易にし、従来不具合解析に要していた時間を短縮することができ、プログラムの開発コストや開発期間などを低減することができる。

【図面の簡単な説明】

【図 1】

本発明に係る情報処理装置の一実施例を示す概略図である。

【図 2】

本発明に係る情報処理装置に搭載される不具合解析プログラムの処理例を示す流れ図である。

【図 3】

本発明の情報処理装置が利用する検証データのフォーマット例を示す概略図である。

【図 4】

本発明の情報処理装置が利用する比較データのフォーマット例を示す概略図である。

【図 5】

本発明の情報処理装置が利用する判定結果データ 1 のフォーマット例を示す概略図である。

【図 6】

本発明の情報処理装置が利用する判定結果データ 2 のフォーマット例を示す概略図である。

【図 7】

本発明の情報処理装置が利用する不具合解決用質問データのフォーマット例を示す概略図である。

【図 8】

第 1 実施例で想定したアプリケーション・プログラムの仕様に沿った動作状況を示す入出力関連図である。

【図 9】

第 1 実施例で想定したアプリケーション・プログラムの不具合による動作状況を示す入出力関連図である。

【図 1 0】

図 2 で示した不具合解析プログラムの処理のうち、ユーザーインターフェース部の詳細な処理例を示す流れ図である。

【図 1 1】

図 1 0 で示したプログラムの動作状況をユーザーに提示した表示例を示すイメージ図である。

【図 1 2】

図 2 で示した不具合解析プログラムの処理のうち、要求解析部の詳細な処理例を示す流れ図である。

【図 1 3】

図 1 0 で示したプログラムの動作状況をユーザーに提示した表示例示すイメージ図である。

【図 1 4】

図 2 で示した不具合解析プログラムの処理のうち、タスク動作判定部の詳細な処理例を示す流れ図である。

【図 1 5】

図 1 0 で示したプログラムの動作状況をユーザーに提示した表示例示すイメージ図である。

【図 1 6】

本発明の情報処理装置が利用するシステムコール対応テーブルの一例を示す概略図である。

【図 1 7】

第 1 実施例における各種データの状態を示した図である。

【図 1 8】

第 1 実施例における各種データの状態を示した図である。

【図 1 9】

第 2 実施例で想定したアプリケーション・プログラムの仕様に沿った動作状況を示す入出力関連図である。

【図 2 0】

第 1 実施例で想定したアプリケーション・プログラムの不具合による動作状況を示す入出力関連図である。

【図 2 1】

第 2 実施例における各種データの状態を示した図である。

【図 2 2】

第 2 実施例における各種データの状態を示した図である。

【図 2 3】

第 2 実施例における各種データの状態を示した図である。

【図 2 4】

第 2 実施例における各種データの状態を示した図である。

【図 2 5】

第 2 実施例における各種データの状態を示した図である。

【図 2 6】

第 2 実施例における各種データの状態を示した図である。

【図 2 7】

従来のデバッガの構成を示した概略図である。

【図 2 8】

従来のデバッガが出力する実行履歴情報の構成を示した概略図である。

【図 2 9】

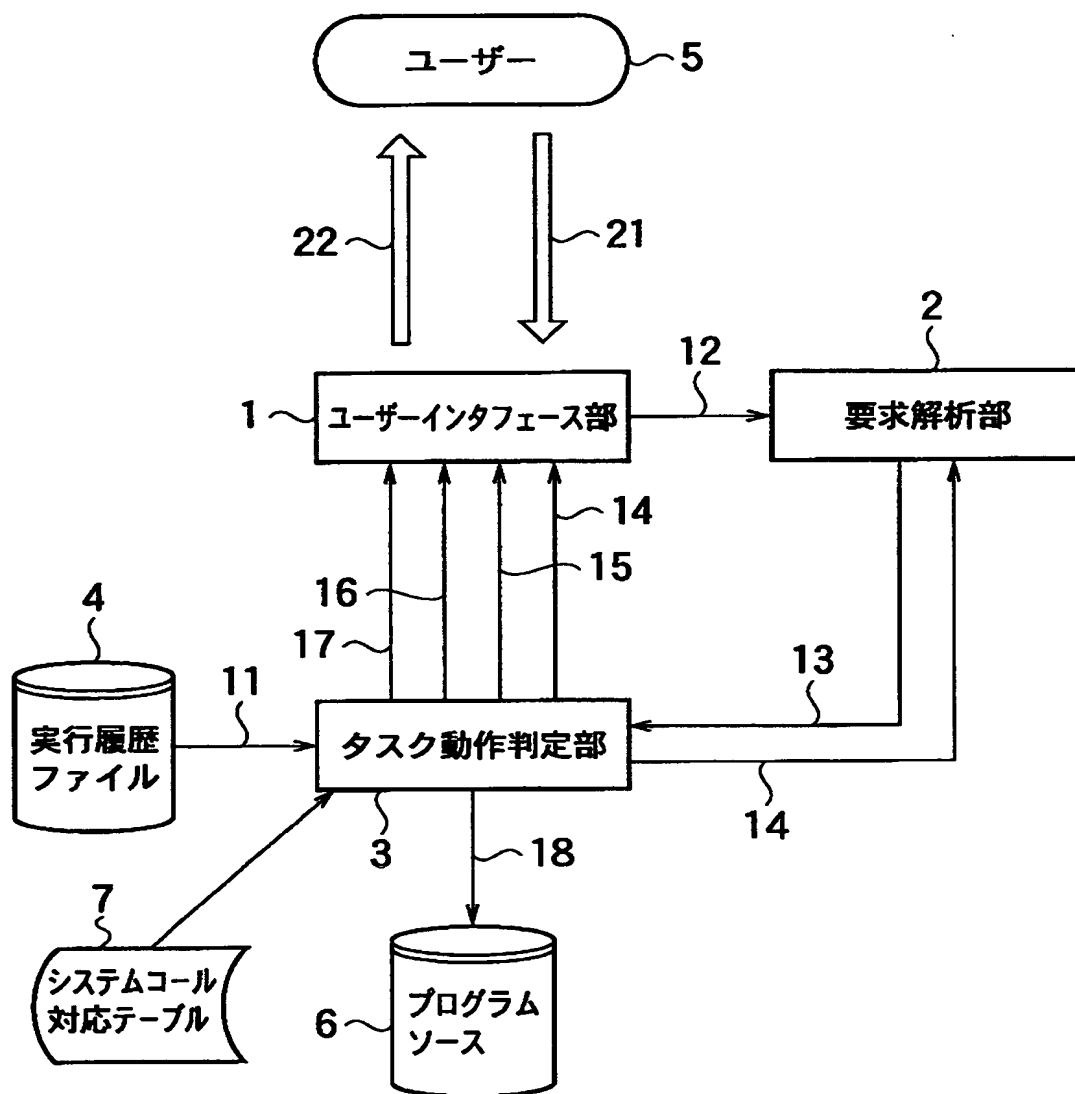
従来の実行履歴情報の構成とデータの一例を示したイメージ図である。

【符号の説明】

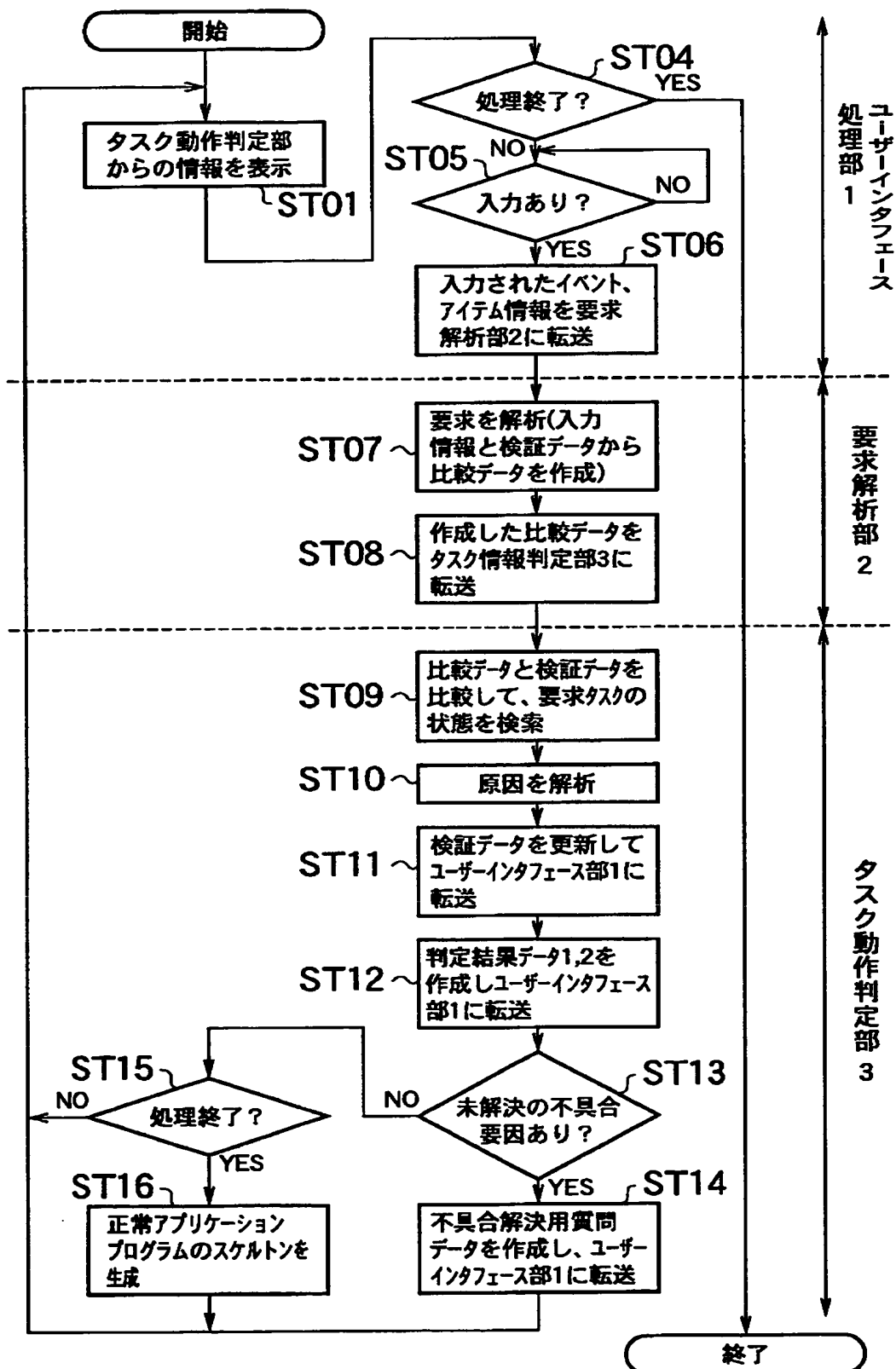
- 1 .... ユーザーインタフェース部
- 2 .... 要求解析部
- 3 .... タスク動作判定部
- 4 .... 実行履歴ファイル
- 5 .... ユーザー
- 6 .... システムコール対応テーブル
- 1 1 .... 実行履歴情報
- 1 2 .... 入力データ
- 1 3 .... 比較データ
- 1 4 .... 検証データ
- 1 5 .... 判定結果データ 1
- 1 6 .... 判定結果データ 2
- 1 7 .... 不具合質問用データ
- 1 8 .... プログラム
- 1 0 1 .... OS
- 1 0 2 .... デバッガ

【書類名】 図面

【図 1】



【図 2】



【図 3】

イベント順	イベント属性 (システムコールの 発行)	発行 システムコール	発行元 タスクID	発行元タスク 優先度	発行元 タスク後の状態	発行先タスク 優先度 (発行先ID)	発行先 タスク後の状態
イベント順	イベント属性 (ディスパッチ)	ディスパッチ元 タスクID	ディスパッチ元 タスク優先度	ディスパッチ先 タスクID	ディスパッチ先 タスク優先度	-	-
イベント順	イベント属性 (割り込み 処理)	ハンドラ属性 (周期起動 ハンドラ、アラーム ハンドラ、割り 込みハンドラ)	ハンドラNo	-	-	-	-
・ ・ ・	・ ・ ・	・ ・ ・	・ ・ ・	・ ・ ・	・ ・ ・	・ ・ ・	・ ・ ・

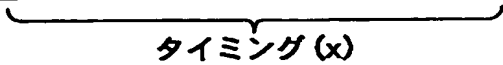
イベント属性：システムコールの発行  
ディスパッチ割り込み処理

ハンドラ属性：周期起動ハンドラ  
アラームハンドラ  
割り込みハンドラ



【図 4】

N	先イベント	後イベント	アイテム(タスク (y))
	:	:	:


  
 タイミング (x)

【図 5】

① N	② 要求タスク	③ 要求タスクの状態
:	:	:
:	:	:

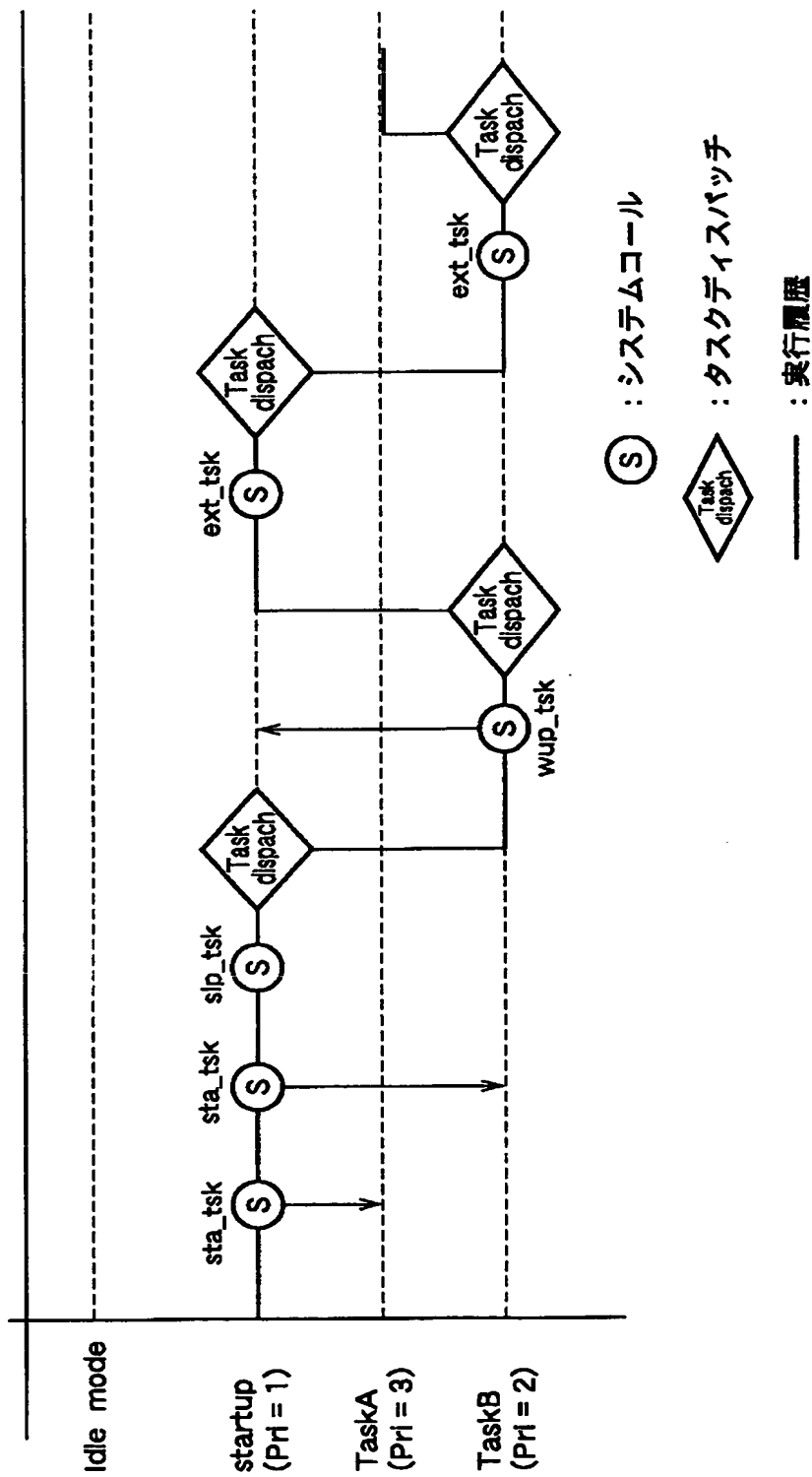
【図 6】

① N	② イベントの最後に実行状態のタスク	③ ext_tsk
:	:	:
:	:	:

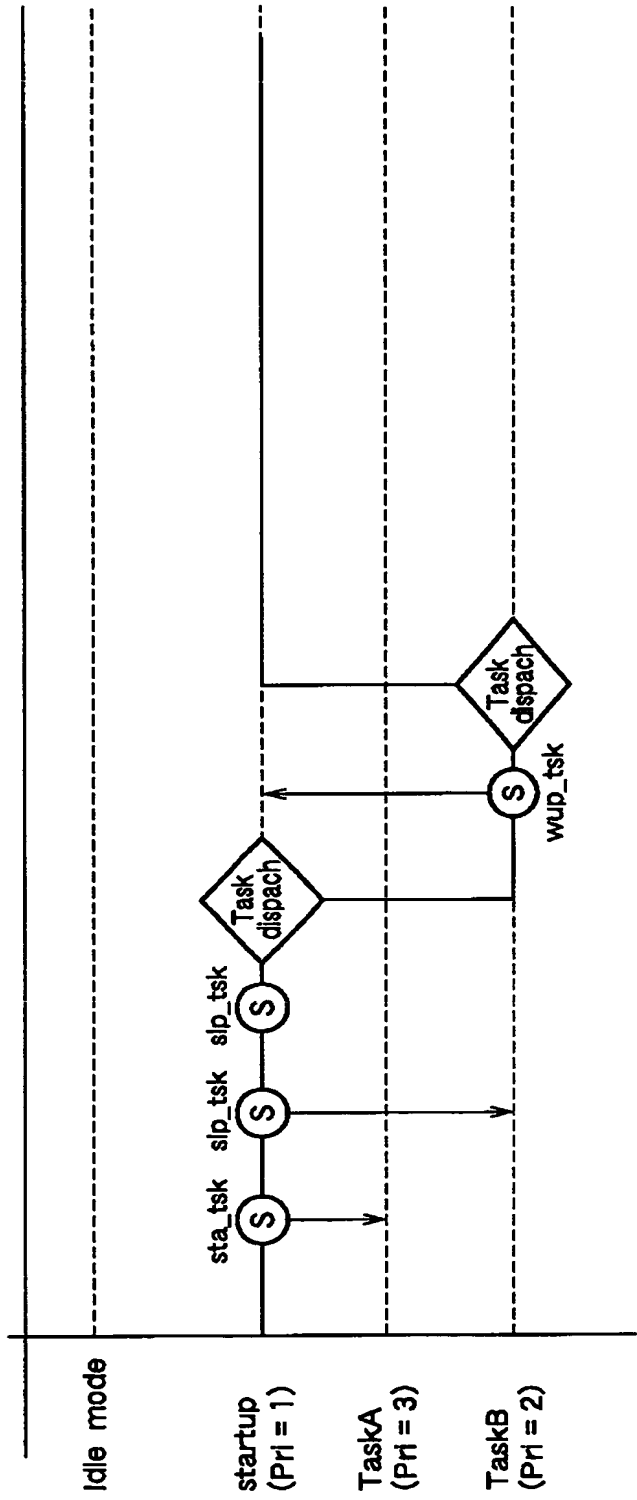
【図 7】

① N	② 要求タスクを実行可能状態にする システムコールの発行対象	③ 要求タスクを実行可能状態にする システムコール
:	:	:
:	:	:

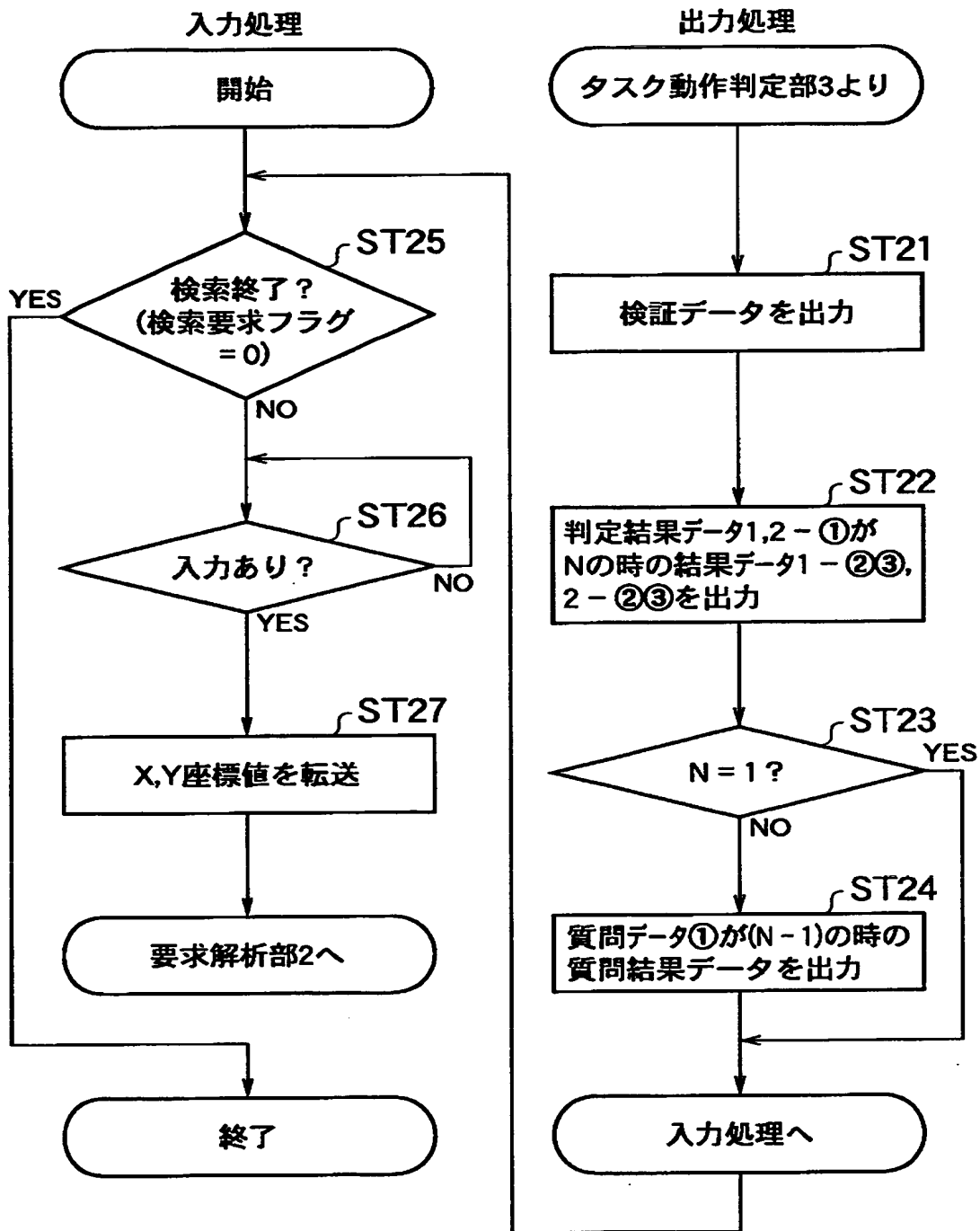
【図 8】



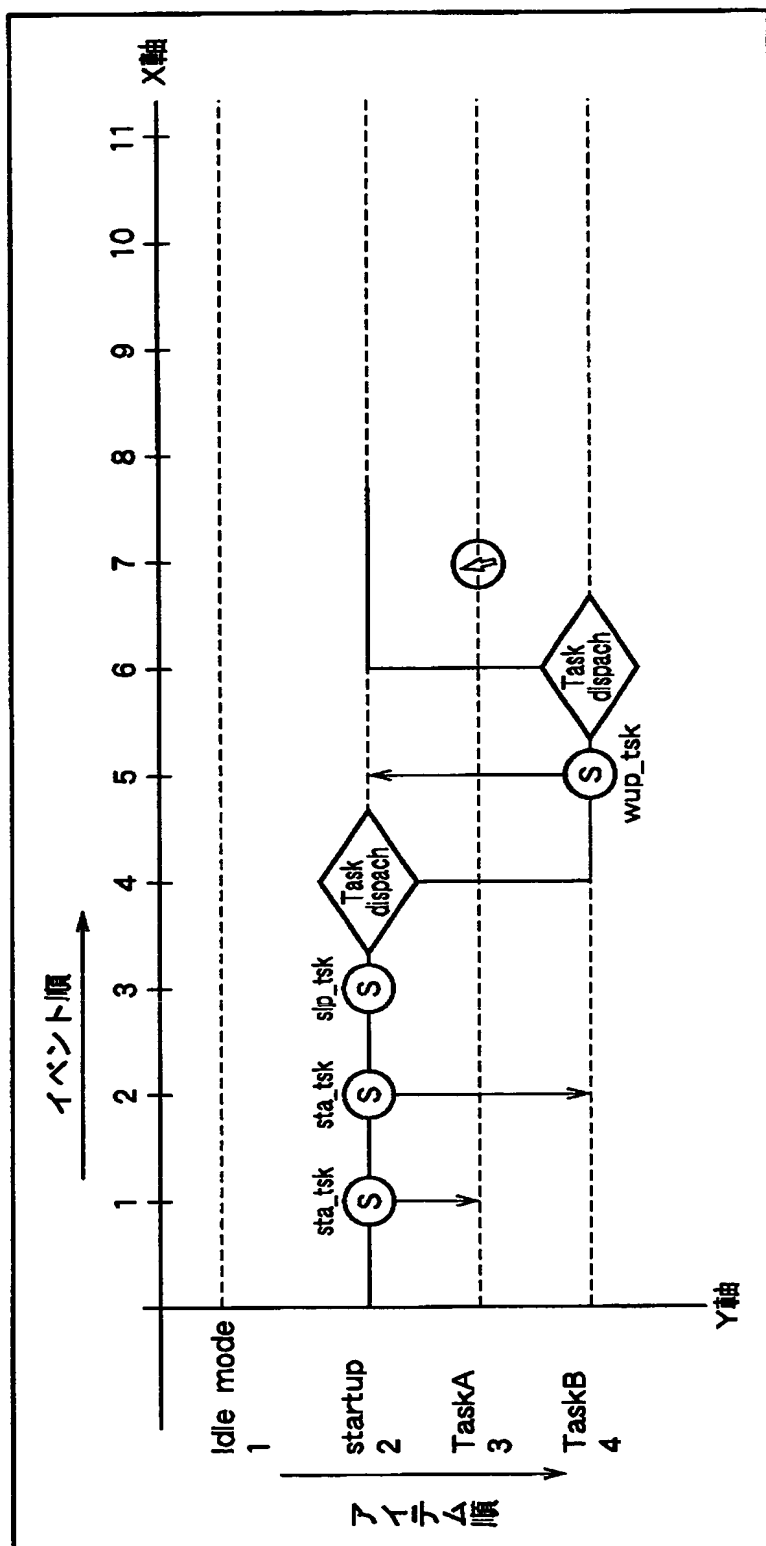
【図 9】



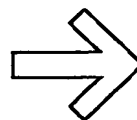
【図 1 0】



【図 1 1】



**入力条件：実行履歴を表す線上市が指定できない**

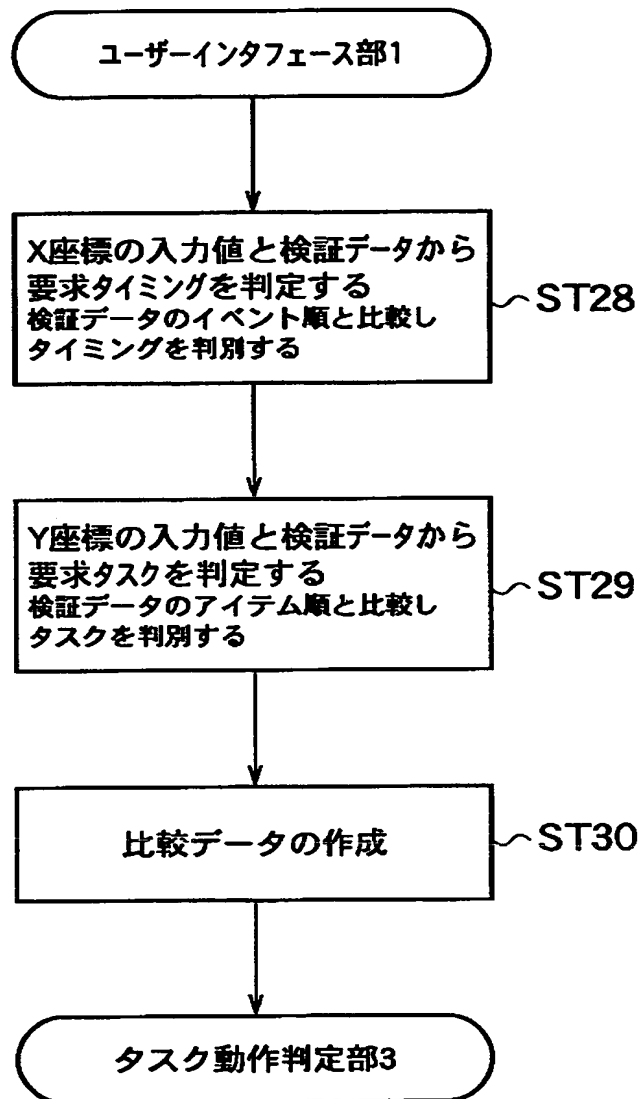


④ : マウスカーソル

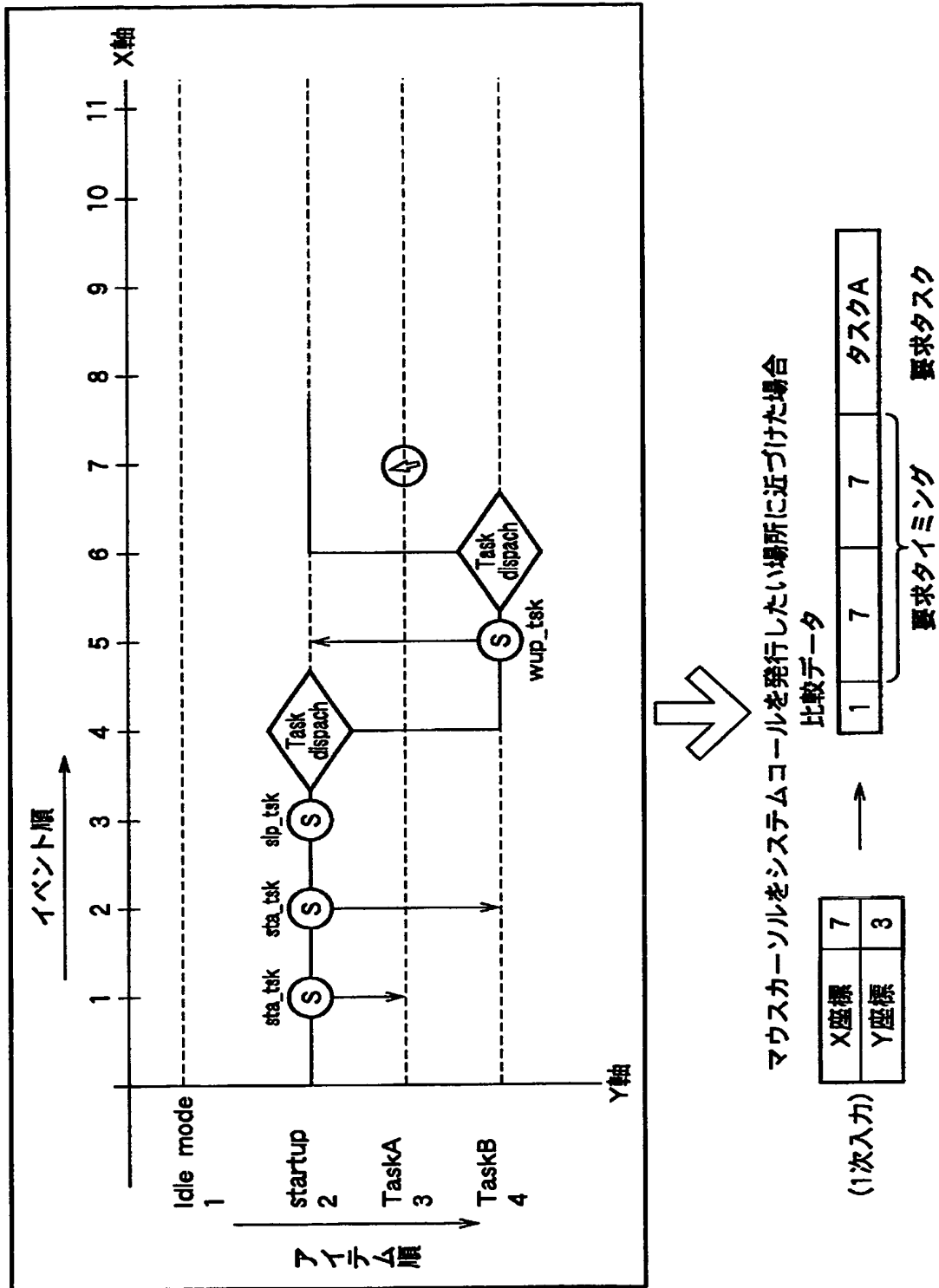
マウスカーソルをシステムコールを発行したい場所に近づける

X座標	7
Y座標	3

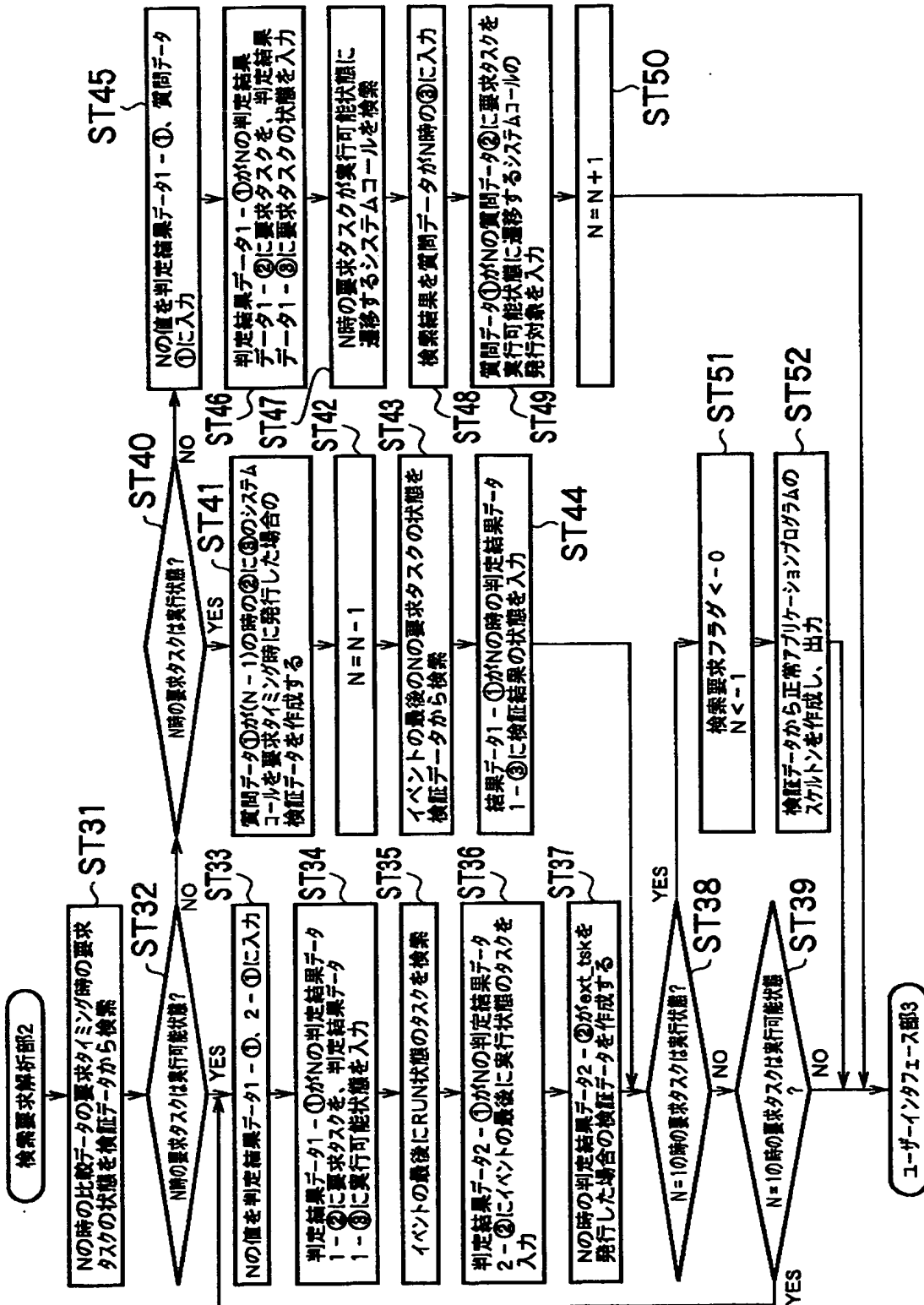
【図 1 2】



【図 1 3】



【図 1 4】





【図 1 5】

<システムコール対応テーブル>

No	システムコール	対応するシステムコール
1	sta_tsk	ext_tsk
2	slp_tsk	wup_tsk
3	wai_sem	sig_sem
⋮	⋮	⋮

【図 1 6】

●【処理1】→【第1次要求】

<検証データ>

イベント順	イベント属性	発行システムコール	ハンドラ属性	ハンドラNO	発行元タスクID	発行元タスク優先度	新タスクの新タスクID	発行元タスクID(発行先資源)	発行元タスク優先度(発行先ID)	新タスクの新タスクID
1	System Call	sta_tsk	-	-	startup	1	実行中	taskA	3	実行可能
2	System Call	sta_tsk	-	-	startup	1	実行中	taskB	2	実行可能
3	System Call	slp_tsk	-	-	startup	1	待ち	-	-	-
4	Task Dispatch	-	-	-	startup	1	-	taskB	2	実行中
5	System Call	wup_tsk	-	-	taskB	2	実行可能	startup	1	実行可能
6	Task Dispatch	-	-	-	taskB	2	-	startup	1	実行中

<比較データ>

N	先イベント	後イベント	アイテム(タスク(y))
1	7	7	taskA

★

<判定結果データ1>

① N	② 要求タスク	③ 要求タスクの状態

<判定結果データ2>

① N	④ イベントの種別と実行状態のタスク	⑤ ext_tsk

<不具合解決用質問データ>

① N	⑥ 要求タスクを実行可能にするシステムコールの発行状態	⑦ 要求タスクを実行可能状態にするシステムコール

【図 1 7】

●[処理2]→[処理終了]

<検証データ>

イベント順	イベント属性	発行システムコール	ハンドラ属性	ハンドラNO	発行元タスクID	発行元タスク優先度	発行タスクの発行後のタスク状態	発行先タスクID(発行先資源)	発行先タスク優先度(発行先ID)	発行先タスクの発行後のタスク状態
1	System Call	sta_tsk	-	-	startup	1	実行中	taskA	3	実行可能
2	System Call	sta_tsk	-	-	startup	1	実行中	taskB	2	実行可能
3	System Call	slp_tsk	-	-	startup	1	待ち	-	-	-
4	Task Dispatch	-	-	-	startup	1	-	taskB	2	実行中
5	System Call	wup_tsk	-	-	taskB	2	実行中	startup	1	実行可能
6	Task Dispatch	-	-	-	taskB	2	-	startup	1	実行中
7	System Call	ext_tsk	-	-	startup	1	休止	-	-	-
8	Task Dispatch	-	-	-	startup	1	-	taskB	2	実行状態
9	System Call	ext_tsk	-	-	taskB	2	休止	-	-	-
10	Task Dispatch	-	-	-	taskB	2	-	taskA	3	実行状態

<比較データ>

N	先イベント	後イベント	アイテム(タスク(y))
1	7	7	taskA

<判定結果データ1>

① N	② 要求タスク	③ 要求タスクの状態	
1	taskA	実行可能	★
1	taskA	実行可能	★

<判定結果データ2>

① N	② イベントの発生に実行状態のタスク	③ ext_tsk	
1	startup	ext_tsk	★
1	taskB	ext_tsk	★

<不具合解決用質問データ>

① N	② 要求タスクを発生可能にするシステムコールの発行順	③ 要求タスクを実行可能状態にするシステムコール

【図 1 8】

## 実施例 1 正常アプリケーションのスケルトン例

```

/*****
File      : sample.c
Data      : 1999/11/11
Developer : TOSHIBA
Application Skeleton
*****/
#include "itron.h"

#define TASK_ID1      1
#define TASK_ID2      2
#define TASK_ID3      3

TASK startup() :
TASK TaskA() :
TASK TaskB() :

TASK startup() :
{
    ER ercd ;

    ercd = sta_tsk(TASK_ID2,0) ;
    ercd = sta_tsk(TASK_ID3,0) ;
    ercd = slp_tsk() ;

    ext_tsk() ; ----- (a)
}

TASK TaskA()
{
    for( ; ; ){
    }

}

TASK TaskB()
{
    ER ercd ;

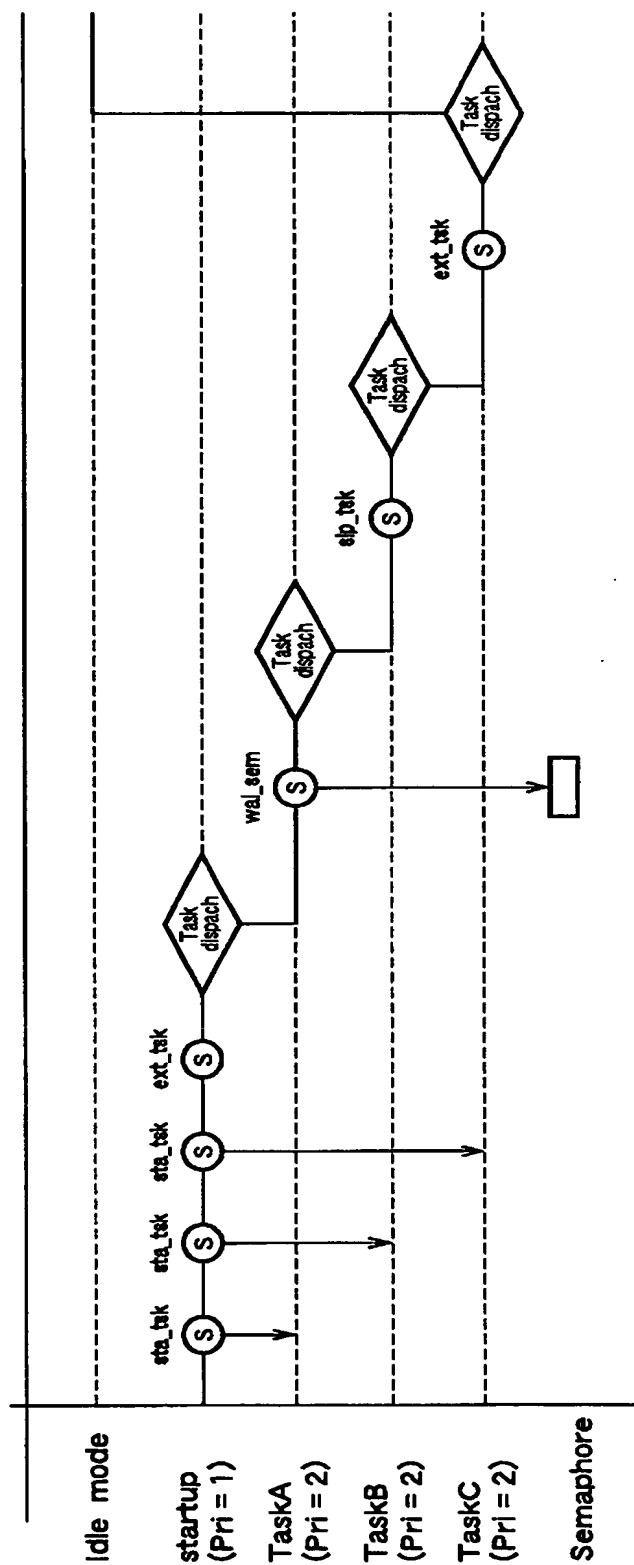
    ercd = wup_tsk(TASK_ID1) ;

    ext_tsk() ; ----- (b)
}

```



【図 20】



【図 2 1】

●【処理1】→【第1次要求】

<検証データ>

イベント順	イベント属性	発行システムコール	ハンドラ属性	ハンドラNO	発行元タスクID	発行元タスク優先度	新タスクの新タスクID	発行先タスクID(発行先資源)	発行先タスク優先度(発行先ID)	発行先タスクの新タスクID
1	System Call	sta_tsk	-	-	startup	1	実行状態	taskA	2	実行可能
2	System Call	sta_tsk	-	-	startup	1	実行状態	taskB	2	実行可能
3	System Call	sta_tsk	-	-	startup	1	実行状態	taskC	2	実行可能
4	System Call	ext_tsk	-	-	startup	1	休止	-	-	-
5	Task Dispatch	-	-	-	startup	1	-	taskA	2	実行状態
6	System Call	wai_sem	-	-	taskA	2	待ち	Semaphore	1	-
7	Task Dispatch	-	-	-	taskA	2	-	taskB	2	実行状態
8	System Call	slp_tsk	-	-	taskB	2	待ち	-	-	-
9	Task Dispatch	-	-	-	taskB	2	-	taskC	2	実行状態
10	System Call	ext_tsk	-	-	taskC	2	休止	-	-	-
11	Task Dispatch	-	-	-	taskC	2	-	Idle Mode	-	-

<比較データ>

N	先イベント	後イベント	アイテム(タスク(y))
1	12	12	taskB ★

<判定結果データ1>

① N	② 要求タスク	③ 要求タスクの状態

<判定結果データ2>

① N	④ イベントの最後の実行状態のタスク	⑤ ext_tsk

<不具合解決用質問データ>

① N	⑥ 要求タスクを実行可能なシステムコールの発行状態	⑦ 要求タスクを実行可能な状態にするシステムコール

【図 2 2】

● [処理2] → [第2次要求]

<検証データ>

イベント順	イベント属性	発行システムコール	ハンドラ属性	ハンドラNO	発行元タスクID	発行元タスク優先度	新タスクの発行後のタスク状態	発行先タスクID(発行先資源)	発行先タスク優先度(発行先ID)	発行先タスクの発行後のタスク状態
1	System Call	sta_tsk	-	-	startup	1	実行状態	taskA	2	実行可能
2	System Call	sta_tsk	-	-	startup	1	実行状態	taskB	2	実行可能
3	System Call	sta_tsk	-	-	startup	1	実行状態	taskC	2	実行可能
4	System Call	ext_tsk	-	-	startup	1	休止	-	-	-
5	Task Dispatch	-	-	-	startup	1	-	taskA	2	実行状態
6	System Call	wai_sem	-	-	taskA	2	待ち	Semaphore	1	-
7	Task Dispatch	-	-	-	taskA	2	-	taskB	2	実行状態
8	System Call	slp_tsk	-	-	taskB	2	待ち	-	-	-
9	Task Dispatch	-	-	-	taskB	2	-	taskC	2	実行状態
10	System Call	ext_tsk	-	-	taskC	2	休止	-	-	-
11	Task Dispatch	-	-	-	taskC	2	-	Idle Mode	-	-

<比較データ>

N	先イベント	後イベント	アイテム(タスク(y))
1	12	12	taskB
2	6	7	taskA

★

<判定結果データ1>

① N	② 要求タスク	③ 要求タスクの状態
1	taskB	待ち

★

<判定結果データ2>

① N	② イベントの発生に実行状態のタスク	③ ext_tsk

<不具合解決用質問データ>

① N	② 要求タスクを実行可能にするシステムコールの発行状態	③ 要求タスクを実行可能状態にするシステムコール
1	taskA	wup_tsk

★



【図 2 3】

●【処理3】→【第3次要求】

<検証データ>

イベント順	イベント属性	発行システムコール	ハンドラ属性	ハンドラNO	発行元タスクID	発行元タスク優先度	発行元タスクの発行後のタスク状態	発行先タスクID(発行先資源)	発行先タスク優先度(発行先ID)	発行先タスクの発行後のタスク状態
1	System Call	sta_tsk	-	-	startup	1	実行状態	taskA	2	実行可能
2	System Call	sta_tsk	-	-	startup	1	実行状態	taskB	2	実行可能
3	System Call	sta_tsk	-	-	startup	1	実行状態	taskC	2	実行可能
4	System Call	ext_tsk	-	-	startup	1	休止	-	-	-
5	Task Dispatch	-	-	-	startup	1	-	taskA	2	実行状態
6	System Call	wai_sem	-	-	taskA	2	待ち	Semaphore	1	-
7	Task Dispatch	-	-	-	taskA	2	-	taskB	2	実行状態
8	System Call	slp_tsk	-	-	taskB	2	待ち	-	-	-
9	Task Dispatch	-	-	-	taskB	2	-	taskC	-	実行状態
10	System Call	ext_tsk	-	-	taskC	2	休止	-	-	-
11	Task Dispatch	-	-	-	taskC	2	-	Idle Mode	-	-

<比較データ>

N	先イベント	後イベント	アイテム(タスク(y))
1	12	12	taskB
2	6	7	taskA
3	9	10	taskC

★

<判定結果データ1>

① N	② 要求タスク	③ 要求タスクの状態
1	taskB	待ち
2	taskA	待ち

★

<判定結果データ2>

① N	④ イベントの直後に実行状態のタスク	⑤ ext_tsk

<不具合解決用質問データ>

① N	⑥ 要求タスクを実行可能にするシステムコールの発行直後	⑦ 要求タスクを実行可能状態にするシステムコール
1	taskA	wup_tsk
2	taskC	sig_sem

★

【図 2 4】

●【処理4】→【第4次要求】

<検証データ>

イベント順	イベント属性	発行システムコール	ハンドラ属性	ハンドラNO	発行元タスクID	発行元タスク優先度	発行元タスクの発行済のタスク状態	発行先タスクID(発行先資源)	発行先タスク優先度(発行先ID)	発行先タスクの発行済のタスク状態
1	System Call	sta_tsk	-	-	startup	1	実行状態	taskA	2	実行可能
2	System Call	sta_tsk	-	-	startup	1	実行状態	taskB	2	実行可能
3	System Call	sta_tsk	-	-	startup	1	実行状態	taskC	2	実行可能
4	System Call	ext_tsk	-	-	startup	1	休止	-	-	-
5	Task Dispatch	-	-	-	startup	1	-	taskA	2	実行状態
6	System Call	wai_sem	-	-	taskA	2	待ち	Semaphore	1	-
7	Task Dispatch	-	-	-	taskA	2	-	taskB	2	実行状態
8	System Call	slp_tsk	-	-	taskB	2	待ち	-	-	-
9	Task Dispatch	-	-	-	taskB	2	-	taskC	2	実行状態
10	System Call	sig_sem	-	-	taskC	2	実行状態	Semaphore	1	-
11	System Call	ext_tsk	-	-	taskC	2	休止	-	-	-
12	Task Dispatch	-	-	-	taskC	2	-	taskA	2	実行状態

<比較データ>

N	先イベント	後イベント	アイテム(タスク(y))
1	12	12	taskB
2	6	7	taskA
3	9	10	taskC

<判定結果データ1>

① N	② 要求タスク	③ 要求タスクの状態
1	taskB	待ち
2	taskA	実行状態

<判定結果データ2>

① N	② イベントの最後の実行状態のタスク	③ ext_tsk

<不具合解決用質問データ>

① N	② 要求タスクを実行可能にするシステムコール発行手段	③ 要求タスクを実行可能状態にするシステムコール
1	taskA	wup_tsk
2	taskC	sig_sem

【図 2 5】

●【処理5】→【第5次要求】

<検証データ>

イベント順	イベント属性	発行システムコール	ハンドラ属性	ハンドラNO	発行元タスクID	発行元タスク優先度	発行元タスクの発行済のタスク状態	発行先タスクID(発行先資源)	発行先タスクの優先度(発行先ID)	発行先タスクの発行済のタスク状態
1	System Call	sta_tsk	-	-	startup	1	実行状態	taskA	2	実行可能
2	System Call	sta_tsk	-	-	startup	1	実行状態	taskB	2	実行可能
3	System Call	sta_tsk	-	-	startup	1	実行状態	taskC	2	実行可能
4	System Call	ext_tsk	-	-	startup	1	休止	-	-	-
5	Task Dispatch	-	-	-	startup	1	-	taskA	2	実行状態
6	System Call	wai_sem	-	-	taskA	2	待ち	Semaphore	1	-
7	Task Dispatch	-	-	-	taskA	2	-	taskB	2	実行状態
8	System Call	slp_tsk	-	-	taskB	2	待ち	-	-	-
9	Task Dispatch	-	-	-	taskB	2	-	taskC	2	実行状態
10	System Call	sig_sem	-	-	taskC	2	実行状態	Semaphore	1	-
11	System Call	ext_tsk	-	-	taskC	2	休止	-	-	-
12	Task Dispatch	-	-	-	taskC	2	-	taskA	2	実行状態
13	System Call	wup_tsk	-	-	taskA	2	実行状態	taskB	2	実行可能★
14	System Call	ext_tsk	-	-	taskA	2	休止	-	-	-
15	Task Dispatch	-	-	-	taskA	2	-	taskB	2	実行状態

<比較データ>

N	先イベント	後イベント	アイテム(タスク(y))
1	12	12	taskB
2	6	7	taskA
3	9	10	taskC
1	13	13	taskB★
2	11	12	taskA

<判定結果データ1>

① N	② 要求タスク	③ 要求タスクの状態
1	taskB	実行状態★
2	taskA	実行状態

<判定結果データ2>

① N	② イベントの最後に実行状態のタスク	③ ext_tsk
1	taskA	ext_tsk★

<不具合解決用質問データ>

① N	② 要求タスクを実行可能にするシステムコールの発行対象	③ 要求タスクを実行可能状態にするシステムコール
1	taskA	wup_tsk
2	taskC	sig_sem

【図 2 6】

●[処理6]→[処理終了]

<検証データ>

イベント順	イベント属性	発行システムコール	ハンドラ属性	ハンドラNO	発行元タスクID	発行元タスク優先度	新タスクの新タスクID(発行先タスク)	発行先タスクID(発行先タスク)	発行先タスク優先度(発行先タスク)	発行先タスクの発行元タスクID
1	System Call	sta_tsk	-	-	startup	1	実行状態	taskA	2	実行可能
2	System Call	sta_tsk	-	-	startup	1	実行状態	taskB	2	実行可能
3	System Call	sta_tsk	-	-	startup	1	実行状態	taskC	2	実行可能
4	System Call	ext_tsk	-	-	startup	1	休止	-	-	-
5	Task Dispatch	-	-	-	startup	1	-	taskA	2	実行状態
6	System Call	wai_sem	-	-	taskA	2	待ち	Semaphore	1	-
7	Task Dispatch	-	-	-	taskA	2	-	taskB	2	実行状態
8	System Call	slp_tsk	-	-	taskB	2	待ち	-	-	-
9	Task Dispatch	-	-	-	taskB	2	-	taskC	2	実行状態
10	System Call	sig_sem	-	-	taskC	2	実行状態	Semaphore	1	-
11	System Call	ext_tsk	-	-	taskC	2	休止	-	-	-
12	Task Dispatch	-	-	-	taskC	2	-	taskA	2	実行状態
13	System Call	wup_tsk	-	-	taskA	2	実行状態	taskB	2	実行可能
14	System Call	ext_tsk	-	-	taskA	2	休止	-	-	-
15	Task Dispatch	-	-	-	taskA	2	-	taskB	2	実行状態

★  
★

<比較データ>

N	先イベント	後イベント	アイテム(タスク(y))
1	12	12	taskB
2	6	7	taskA
3	9	10	taskC
1	13	13	taskB
2	6	7	taskA

<判定結果データ1>

① N	② 要求タスク	③ 要求タスクの状態
1	taskB	実行可能
2	taskA	実行状態

★

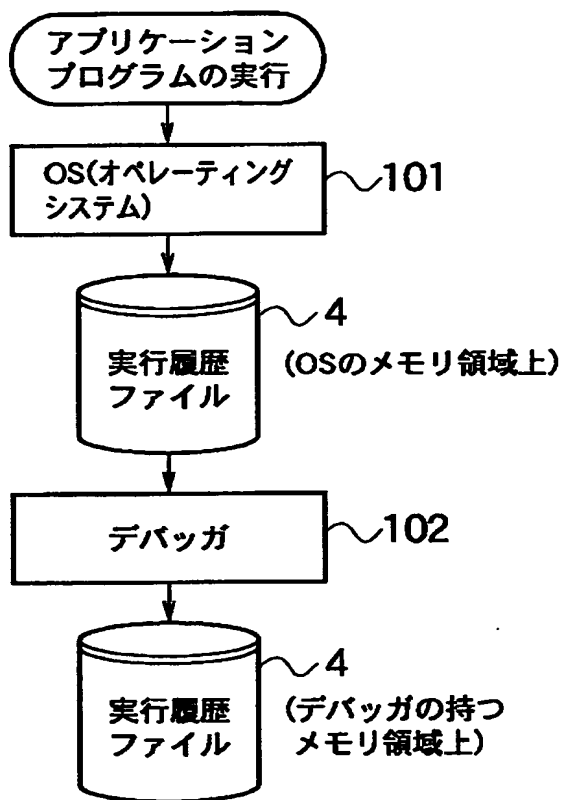
<判定結果データ2>

① N	④ イベントの最後の実行状態のタスク	⑤ ext_tsk
1	taskA	ext_tsk

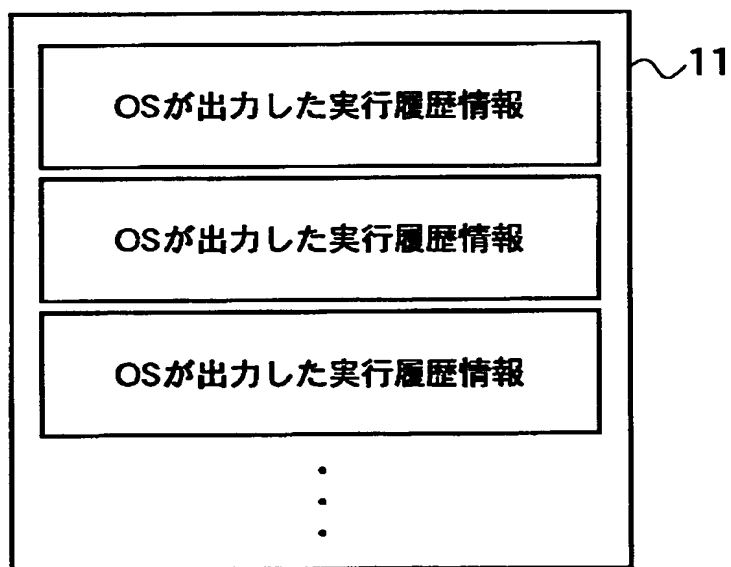
<不具合解決用質問データ>

① N	⑥ 要求タスクを実行可能にするシステムコールの発行対象	⑦ 要求タスクを実行可能状態にするシステムコール
1	taskA	wup_tsk
2	taskC	sig_sem

【図 2 7】



【図 2 8】



【図 2 9】

(a)

type	oid	sysid	obj
------	-----	-------	-----

(b)

6	0	1		...(イ)
1	1	-9	2	...(ロ)
1	1	-9	3	...(ハ)
1	1	-17		...(ニ)
6	1	3		...(ホ)
1	3	-19	1	...(ヘ)
6	3	1		...(ト)
	.			
	.			
	.			

但し、  
 sysid : sta\_txt .... -9  
       : ext\_txt .... -10  
       : slp\_txt .... -17  
       : wup\_txt .... -19  
 とする。

【書類名】 要約書

【要約】

【課題】 対話形式でユーザーから指摘された不具合箇所とプログラムの実行履歴情報から不具合要因と解決策を特定し、これをユーザーに提示することができ、また、特定した不具合要因と解決策から、該不具合を解決したプログラムを生成してユーザーに提示することができる情報処理装置を提供すること。

【解決手段】 プログラムの実行履歴情報をもとに、該プログラムの動作状況を時系列にユーザーに表示する表示手段と、表示された動作状況の中でユーザーが不具合の箇所を指定するための入力手段と、前記入力手段によってユーザーから指定された不具合の箇所と該プログラムの動作状況から該不具合要因を解析し、この不具合要因を解決するための解決策を特定する動作解析手段と、を有し、前記動作解析手段は、特定した解決策を反映させた前記動作状況を再作成し、前記表示手段は、前記不具合要因と前記解決策と再作成した動作状況とをユーザーに表示する。

【選択図】 図 1

出 願 人 履 歴 情 報

識別番号 [ 5 9 8 0 1 0 5 6 2 ]

1. 変更年月日 1 9 9 8 年 1 月 2 3 日

[変更理由] 新規登録

住 所 神奈川県川崎市幸区堀川町 5 8 0 番地

氏 名 東芝エルエスアイシステムサポート株式会社



出 願 人 履 歴 情 報

識別番号 [ 0 0 0 0 0 3 0 7 8 ]

1. 変更年月日 1 9 9 0 年 8 月 2 2 日  
[変更理由] 新規登録  
住 所 神奈川県川崎市幸区堀川町 7 2 番地  
氏 名 株式会社東芝